反復深化 A* 探索によるもっともらしいプログラムの効率的な生成

網代 育大† 上田 和紀

早稲田大学 理工学部 情報学科

(†現所属: 日本電気株式会社)

{ajiro,ueda}@ueda.info.waseda.ac.jp

概要

1. なぜ「もっともらしいプログラム」を生成するのか？
2. 並行論理型言語の紹介
3. モード・型解析によるプログラム検証
4. ヒューリスティクスによるもっともらしいプログラムの選別
5. 並行論理プログラム自動修正系 Kima
6. もっともらしさに関する反復深化 A* 探索による最適化
7. 実験結果

背景と動機: なぜもっともらしいプログラムを生成するのか？

- 正しいプログラムを書くのは難しいが、だいたい正しいプログラムを書くのは比較的容易である。
- これまで「保証に」使われていたプログラム解析技術を「攻撃的に」利用することにより、プログラムの正しい形状を予測できないか。
- 計算機性能の向上によって、個人の所有する計算機であっても、計算能力を持て余している (cf. Grid)。

余剰計算能力と静的解析技術を利用して、いい加減にプログラムを書くための技術を確立したい。

成果物: 並行論理プログラム自動修正系 Kima

- プログラムに関する明示的な仕様や宣言を与えることなく KL1 プログラムの軽微な誤り (変数の書き誤り) を自動的に修正するシステム Kima を開発した。
- 「もっともらしい」プログラムが満たす構文上の制約があることを突き止めた。
- 修正案の探索の最適化技術を開発して実装した。
  - 「もっともらしさ」に関する反復深化 A* 探索
- 実験により、変数の 1 箇所の誤りに対してはその 90% 以上を自動的に修正できることを示した (2 箇所以上の誤りに対してはそれ以上)。

関連技術

- **仕様記述** (e.g., Z, CafeOBJ)
- **入力予測** UI (e.g., Web browser, PO BOX, T9)
- **遺伝的プログラミング**
- **帰納論理プログラミング** (programming by examples)

もっともらしいプログラムの定義

- もっとも基本的なプログラム解析技術である(並行論理型言語における) 強型/モードの体系を利用
  - 強型体系 (strong typing) はデータ型の一貫性を保証し、多くの簡単な誤りを静的に検出することができる。
  - 強モード体系 (strong moding) はデータフローの一貫性を保証。
- もっともらしいプログラムが満たす構文上の(簡単な)制約をヒューリスティクスとして規定

もっともらしいプログラム = 型・モードづけされ、ヒューリスティクスに合致したプログラム

並行論理プログラミング

並行論理型言語の特徴

- セマンティクスのレベルで並行性を扱っていることにより、並行・並列・分散計算の記述に適している。
- 論理型言語 (Prolog) からバックトラックを除いた言語。
- 計算は (並列計算/論理プログラミングにおける) リダクションによって進む。
- プロセス間通信は、論理変数を通じて暗黙のうちに行なわれる。

KL1 プログラムは以下のような節の集合：

```
p(X,Z) :- true | q(X,Y)@node(0), r(Y,Z)@node(1).
```

ここで デモ

1. Append プログラムの1箇所(深さ1)の誤り
2. 組合せプログラムの独立した2箇所(深さ1)の誤り
3. クイックソートプログラムの2箇所(深さ2)の誤り

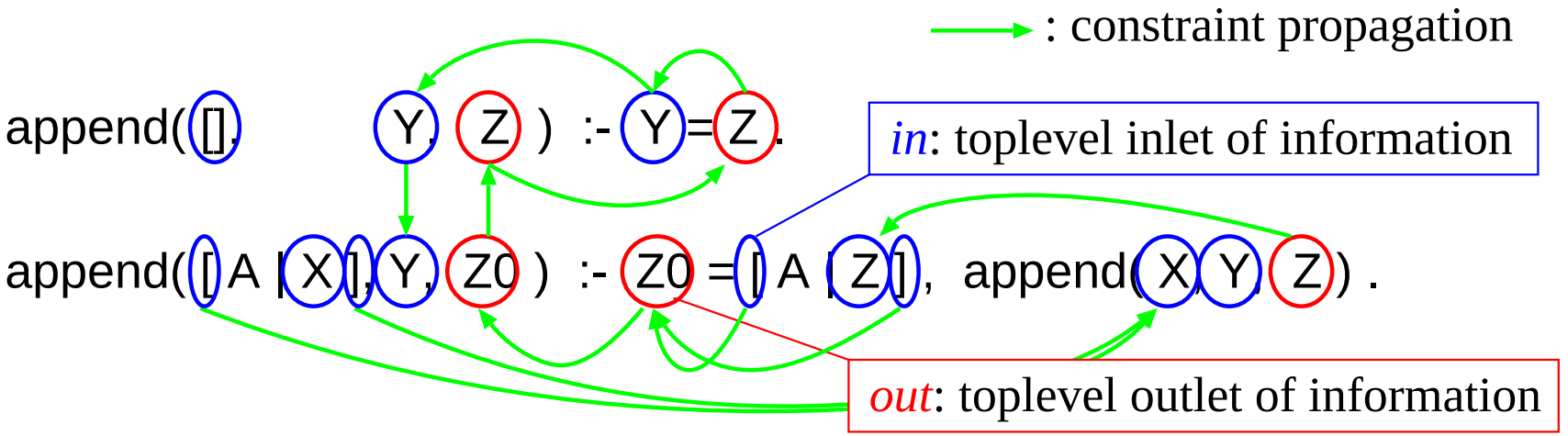
Kima の動作原理

基本アルゴリズム

1. モード/型制約の矛盾する極小部分集合の計算による、誤り箇所候補の特定
2. 矛盾する極小部分集合のグループ化による誤りの分類
3. 以下の手続きの繰返しによる generate-and-test
   (a) 特定した場所の周辺の機械的な書換え
   (b) 書換えたプログラムのモード/型の再計算

- 計算量 = 可能な書換え × モード(型)解析の手間
- モード(型)解析の手間はプログラムサイズに比例

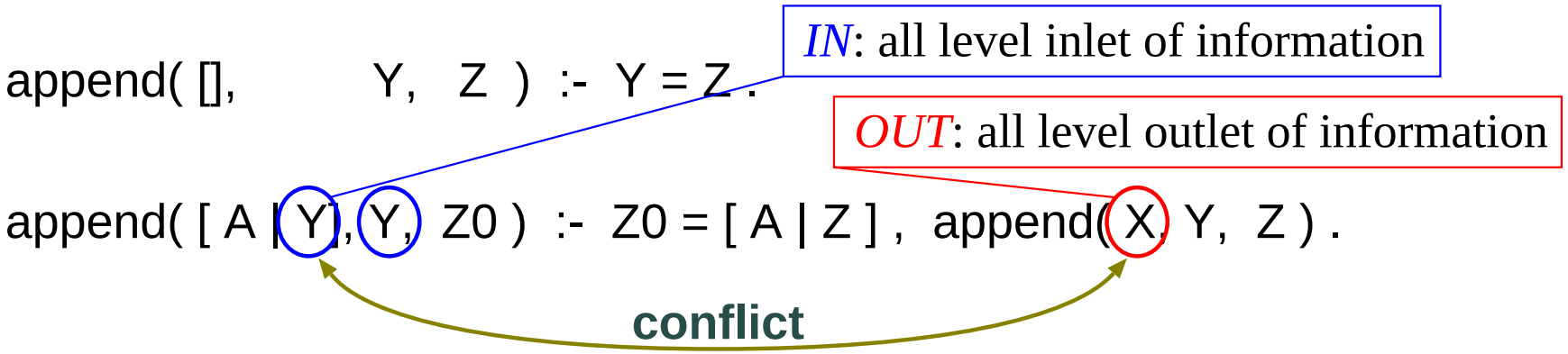
モード解析による情報の流れの解析



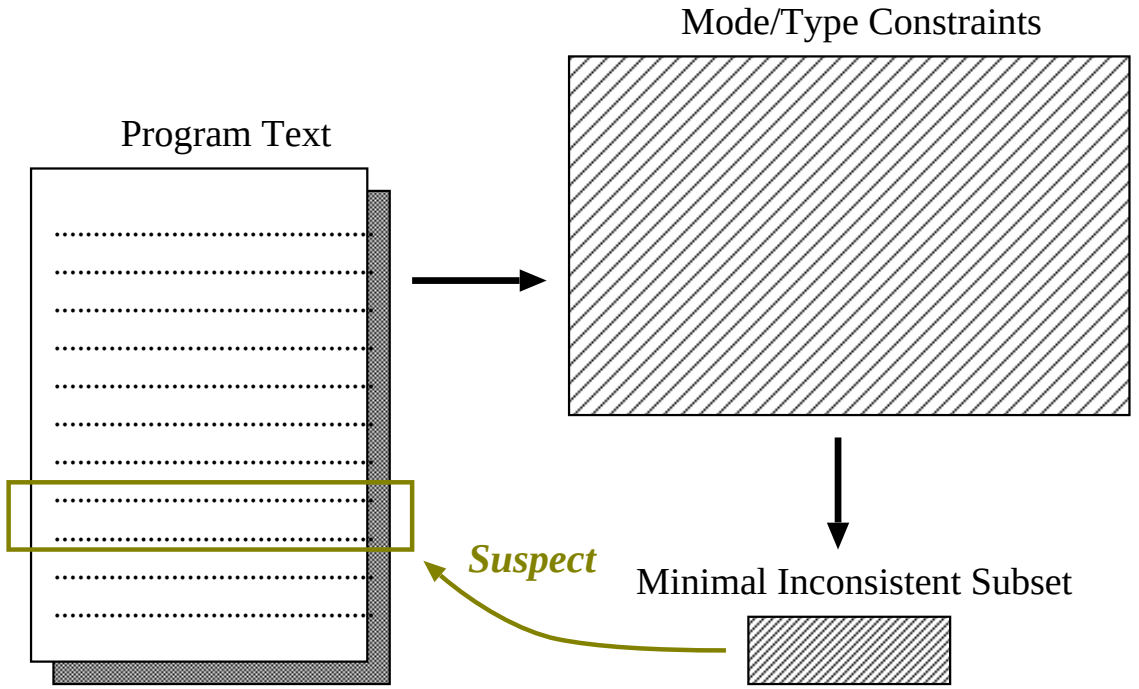
- モード解析は多数の簡単なモード制約の制約充足問題である。
- モード制約は
  - プログラムテキスト中の記号出現と
  - モードづけ規則によって課される。
- データ型に関しても同様の方法で解析可能

モード解析による誤りの検出

- 正しいプログラムはモード/型制約の集合が充足可能 (well-moded/typed) である。
- 誤りを含んだプログラムは、制約集合が充足不可能 (non-well-moded/typed) となることが多い。



矛盾する極小部分集合による誤り箇所の特定



- 矛盾する極小部分集合は間違った制約を含む。
- 極小部分集合サイズの増加は、プログラムサイズの増加に比べて小さい（多くの場合、集合の要素数は 10 以下)。

矛盾する極小部分集合を求めるアルゴリズム

$\{c_1, \ldots, c_n\}$: a multiset of constraints (input),
S: a minimal inconsistent subset (output).

$c_{n+1} \leftarrow \mathit{false}$;
$S \leftarrow \{\}$;
while S is consistent **do**
  $D \leftarrow S$; $i \leftarrow 0$;
  while D is consistent **do**
    $i \leftarrow i + 1$; $D \leftarrow D \cup \{c_i\}$
  end while;
  $S \leftarrow S \cup \{c_i\}$
end while;
if $i = n + 1$ **then** $S \leftarrow \{\}$ **fi**

- **計算量はプログラムサイズに対してほぼ線形オーダー。**

ヒューリスティクス: もっともらしい変数の出現の仕方

優先度づけ Kima は節中における変数の出現の仕方によって修正案に優先度をつける。

- 変数は一対一通信に使われるのがもっともらしい。
- もっともらしい 2回 $>$ 3回以上 $\gg$ 1回もっともらしくない

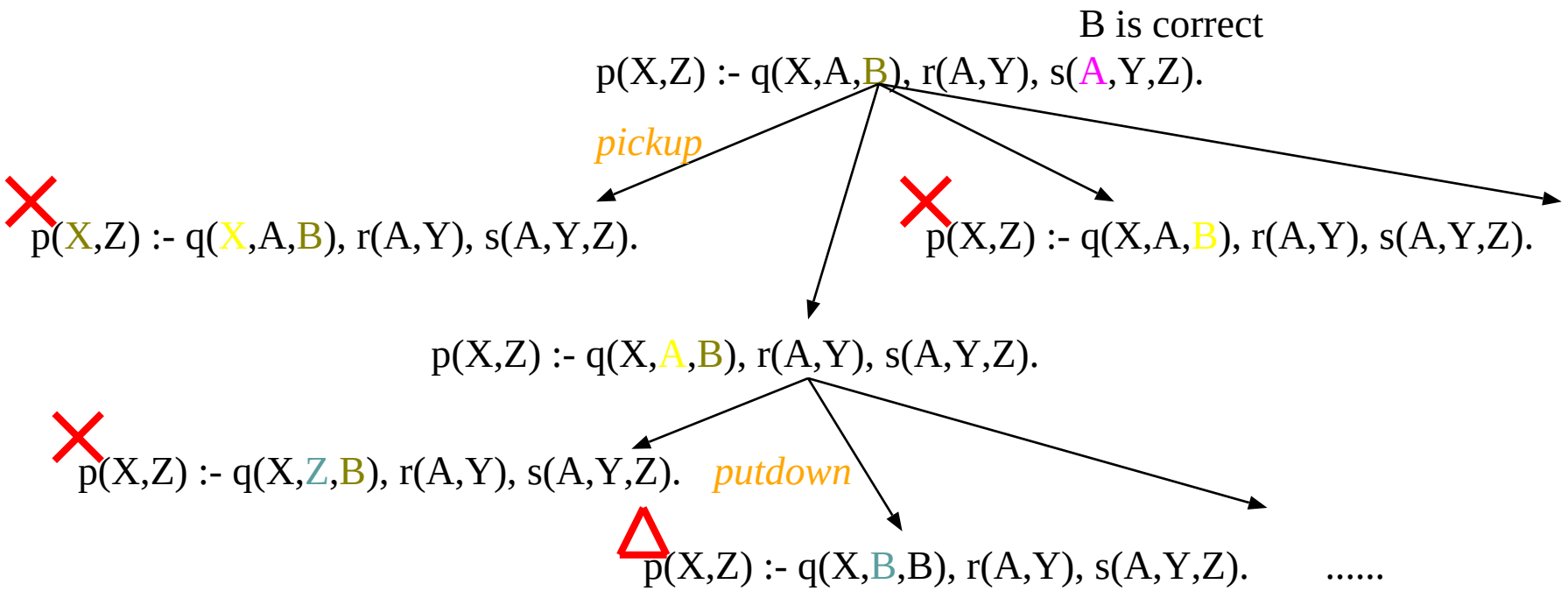
検出規則 Kima は節中に1回しか現れない変数の変数名が下線 “_” で始まっていないとき、これを誤りとして検出できる。

この検査には書き換えた節中の変数の出現数を数えるだけで良い。

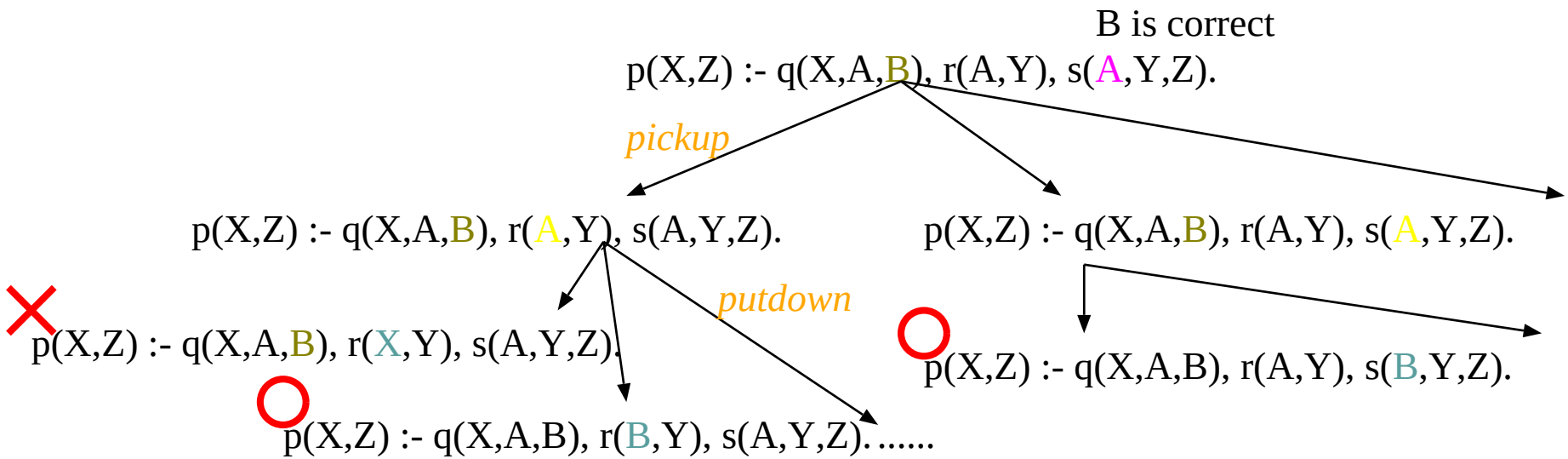
モード/型の再計算には、プログラム全体の解析が必要。

もっともらしさに関する反復深化 A* 探索(1/2)

- 変数 B が疑われており、深さ1の探索を行なうとする。
- 初期要求: 10 変数出現 = 5 変数 × 2 出現
- 1 mutation = 1 pickup + 1 putdown



もっともらしさに関する反復深化 A* 探索(2/2)



- 要求を満たす書換えについて、モード・型の検査を行なう
- 修正案が見つからなかった場合は、要求水準を少し下げて再探索
  - 10 変数出現 = 3 変数 × 2 出現 + 1 変数 × 4 出現
    または 2 変数 × 2 出現 + 2 変数 × 3 出現

実験1: もっともらしいプログラムの分布

Program	N	Total cases	Plausible programs	With caller info	Related to guard
append	1	58	0	0	0
	2	1200	7	1	0
	3	16980	14	3	0
	4	167842	29	5	0
fibonacci	1	118	11	9	4
	2	4668	66	3	3
	3	133045	309	5	5
quicksort	1	300	9	4	4
	2	12102	33	17	4
	3	337455	76	48	16

- オリジナルの正しいプログラムの周辺に存在するもっともらしいプログラムの数は極めて少ない。

実験2: Kima の平均応答時間

プログラム (行数)	節中の変数出現数の平均	N	検出に成功した事例	解析を中止した事例	提示された修正案の数	応答時間 (秒)
nqueen	10.4	1	96	0	1.10	0.439
(67行)		2	96	1	2.17	19.1
gen-test	11.6	1	99	0	1.67	2.04
(200行)		2	83	3	5.74	16.5
tgraph	17.6	1	100	0	1.21	10.9
(218行)		2	100	37	2.35	60.4
graph	16.8	1	98	1	1.15	21.7
(390行)		2	97	39	2.28	87.4

(実験環境: Linux 2.4.3 + PentiumIII 733 MHz + 768 MB memory)

- 提示される修正案の数は 1 か、ごく少数
- 未最適化版では $N = 2$ の誤りは、ほとんど修正不可能

今後の予定: 自動プログラミングへの応用

入出力例を与えることによって、変数の記述を自動化

```
:- instance qsort(+[5,2,7,3],-[2,3,5,7],+[]).
qsort/3 :- X=[]    | =/2.
qsort/3 :- X=[X|X] |
  part/4,qsort/3,X=[X|X],qsort/3.
part(_,[],    S, L ) :- true | S=[],L=[].
part(A,[X|Xs],S0,L ) :- A>=X | S0=[X|S],part(A,Xs,S,L).
part(A,[X|Xs],S, L0) :- A< X | L0=[X|L],part(A,Xs,S,L).
```

⇓ メタインタプリタの援用

```
qsort([],    Ys0,Ys ) :- true | Ys=Ys0.
qsort([X|Xs],Ys0,Ys3) :- true |
  part(X,Xs,S,L),qsort(S,Ys0,Ys1),Ys1=[X|Ys2],qsort(L,Ys2,Ys3).
```

まとめ

- 静的解析と探索技術を利用して、曖昧なプログラムからもっともらしいプログラムを自動生成する枠組を提案した。
- 提案する枠組を並行論理プログラム自動修正系 Kima として実装した。
- Kima はそれ自身 KL1 で記述されており、現在の大きさは約 5,200 行。以下の HP でフリーソフトとして公開

  http://www.ueda.info.waseda.ac.jp/~ajiro/study-j.html